```
IMPACT{
  { turns, magnitude:multiple, ordering:strict} ;
  % Turn assignment is handled in the game itself; sometimes the govt agent makes a few moves, sometimes it
doesn't

  { roles, speaker, listener} ;

  { players, min:2, max:2 } ;
  { player, id:user } ;
  { player, id:govt } ;

  % the game is between Government (represented by the system) and the user of the system

  { store, id:possibleCurrentStates, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ;  % All the
possible combinations (disjunctions) of start states
  { store, id:possibleDesiredStates, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % All the
possible combinations (disjunctions) of end states
  { store, id:possibleActions, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % all the
possible actions
  { store, id:possibleValues, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % all the possible
values
  { store, id:currentStates, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % all the agreed
current states (needed to determine which action can be performed)
  { store, id:desiredStates, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % all the agreed
desired states (needed to determine which action can be performed)
  { store, id:values, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % all the agreed values
  { store, id:action, owner:shared, structure:set, visibility:public, contents:$userDefined$ } ; % the agreed action
  { store, id:governmentcomm, owner:govt, structure:set, visibility:public, contents:$userDefined$ } ; % the
government's commitments. They are not directly used in this protocol
  { store, id:usercomm, owner:user, structure:set, visibility:public, contents:$userDefined$ } ; % the user's
commitments

  {transforce, {<agree_cs, C>, <agree_ds, D>, <agree_v, V>}, {agree_a, {a}}, {<{C,D,V},{a}>,
Value_Practical_Reasoning_Scheme}}; % this is the transitional force between all the agree moves (which assert the
premises) and the agree action move (which asserts the conclusion)

  { backtrack, on };
  % backtrack is on because the govt agent has to ask a series of questions

  { rule, StartingRule, scope:initial,
    { assign(govt, speaker) & move(add, future, question_cs, S, govt, {inspect(in, S, PossibleCurrentStates)}) }} ;
          % The game starts by adding a question for each possible starting state disjunction {p,q} to govt's moveset
```

```
 {interaction, question_cs, {p,q}, questioning, {p}, questioning, {q}, "Is p or q true in the current situation?",
        { assign(user, speaker) & move(add, next, agree_cs, {p}, user) & move(add, next, agree_cs, {q}, user) }};
        % after govt asks a question about the starting states the user can agree with one of them

 {interaction, agree_cs, {p}, asserting, {p}, "p is true in the current situation",
        { if {size(!empty, LegalMoves, govt)}
          then {assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, CurrentStates)}
          else {assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, CurrentStates) & move(add,
future, question_ds, s, govt, {inspect(in, s, PossibleDesiredStates)})}
        }} ;
        % if the user agrees to some state p and govt's moveset is not empty (i.e. the govt can still ask question_cs
moves), then any agree_cs move is deleted from user's moveset,
        % the state p is added to usercomm and currentStates and turn goes to govt (who can ask another
question_cs). Else (if govt's moveset is empty) the govt starts asking which
        % desired states the user wants.

 {interaction, question_ds, {p,q}, questioning, {p}, questioning, {q}, "By implementing the policy do we want to achieve
p or q?",
        { assign(user, speaker) & move(add, next, agree_ds, {p}, user) & move(add, next, agree_ds, {q}, usercomm)
}};
        % after govt asks a question about a desired state the user can agree with one of them

 {interaction, agree_ds, {p}, asserting, {p}, "We want to achieve p",
        { if {size(!empty, Legalmoves, govt)}
          then {assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, DesiredStates)}
          elseif {extCondition(NotPossible{PossibleActions, CurrentStates, DesiredStates})}
          then {assign(govt, speaker) & move(add, future, no_action, govt)}
          else {assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, DesiredStates) & move(add,
future, question_v, s, govt, {inspect(in, s, PossibleValues)}}} ;
 % if govt's moveset is not empty (i.e. the govt can still ask question_ds moves), then any agree_ds move is deleted
from user's moveset, the state p is added to
 % usercomm and turn goes to govt (who can ask another question_ds). Else if with  CurrentStates and
DesiredStates there is no possible action
 % then the govt can do a no_action move or else the government can ask which possible values the user agrees to
(question_v)

  {interaction, no_action, "There is no possible policy action to get from CurrentStates to DesiredStates, please
choose other things you want to achieve",
        { store{remove, T, DesiredStates} & move(add, future, question_ds, s, {inspect(in, s,
PossibleDesiredStates)})} ;
  % if no actions are possible the DesiredStates store is emptied and the desired state questions are again added to
govt's moveset. The user stays committed to
  % the "impossible" desiredStates.
```

```
 {interaction, question_v, {p}, questioning, {p}, "Which values do we want to promote?",
        { assign(user, speaker) & move(add, next, agree_v, {p}, user) & move(add, next, disagree_v, {p}, user) }};
        % after govt asks a question about values the user can agree or disagree

 {interaction, agree_v, {p}, asserting, {p}, "This will promote p",
        { if {size(!empty, Legalmoves, govt)}
          then {assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, values)}
          else {assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, values) & move(add, future,
question_a, {S}, govt, {inspect(in, s, PossibleActions) & extCondition(Possible{s, CurrentStates, DesiredStates})}}
        }} ;
 % again the govt asks questions until its moveset is empty. Otherwise it is checked which actions are possible and
these are asked.

 {interaction, disagree_v, {p}, disagreeing, {p}, "This will not promote p",
        { if {size(!empty, Legalmoves, govt)}
          then {assign(govt, speaker)}
          else {assign(govt, speaker) & move(add, future, question_a, {S}, govt, {inspect(in, s, PossibleActions) &
extCondition(Possible{s, CurrentStates, DesiredStates})}}
        }} ;

 {interaction, question_a, {S}, questioning, S, "Which action from S should we perform?",
        { assign(user, speaker) & move(add, next, agree_a, s, user) }};
        % the user can agree to one of S, so for each element of S an agree_a move is added

 {interaction, agree_a, {p}, asserting, {p}, "We should perform p",
        { assign(govt, speaker) & store(add, {p}, usercomm) & store(add, {p}, Action) & move(add, future, summary,
govt) }} ;
        % once the user agrees to an action the rest of the agree_a moves are deleted from his moveset and
question_v moves are added to govt's moveset

 {interaction, summary, "According to you, given CurrentStates, Actions will result in DesiredStates, which will
promote Values",
        { status(terminate, IMPACT) }};

 }
}
```